原创 **回读GPU缓存数据 同步VS异步**

肖宏宇

发布时间:2022.08.23 15:30 151 7 0 更多
已推荐到：职业精选-程序/游戏客户端

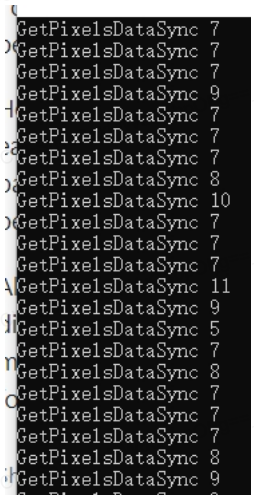
> 在视频录制的过程中，我目前使用的还是软编码，所以需要从GPU读取RT的数据，传递到CPU，这一步之前是使用的同步方式，但是现在发现同步方式太慢了，而且存在诸多弊端，本文就同步和异步方式进行对比和分析，以GL为例。异步方式目前NeoX引擎组已经在Gaea分支上实现，本文只是阐述我自己的思考。

1.同步回读

相比于回读Texture，回读FrameBuffer更简单一些，也能突出本文的重点，所以本文将以回读FrameBuffer为实验主体。那么这就很简单了，如下函数就可以实现同步回读：

```
void GetPixelsDataSync(int render_width, int render_height, unsigned char** data)
{
	auto start = std::chrono::steady_clock::now();

	glPixelStorei(GL_PACK_ALIGNMENT, 1);
	glReadPixels(0, 0, render_width, render_height, GL_RGBA, GL_UNSIGNED_BYTE, *data);
	glPixelStorei(GL_PACK_ALIGNMENT, 4);

	auto finish = std::chrono::steady_clock::now();
	auto ms = std::chrono::duration_cast<std::chrono::milliseconds>(finish - start).count();
	std::cout << "GetPixelsDataSync " << ms << std::endl;
}
```

注意这里的glPixelStorei函数的使用，后文将会做一个实验说明它的必要性。 那么这里面最重要的函数就是glReadPixels，它最后的参数 data 就是用来接受GPU传递到CPU的数据，这是一个同步操作，会占用CPU的时钟周期，非常的慢，测试结果如下：



渲染的尺寸是1920 * 1080，结果这个耗时就要10ms左右，这个在游戏里面是无法接受的。而且，我们看看neox的接口：

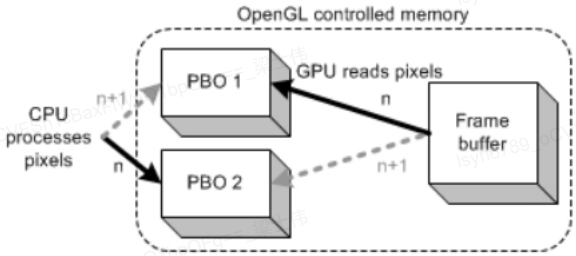
```
void DeviceTextureProxy::GetPixelsData(ReadRequest& request)
{
    DeviceCommandBuffer* commandBuffer = DeviceCommandBuffer::GetInstance();
    DCB_ENCODE_COMMAND_2(
            commandBuffer,     DeviceTextureGetPixelData,
```

```
    commandBuffer->KickAndWaitFinish();
}
```

它这里还有个KickAndWaitFinish，就是要执行完当前CommandBuffer里面全部的指令，这无疑会等待更长的时间，所以同步回读在实时游戏中不建议使用，G92的魔法相机使用了同步方式，必然造成帧率下降，之后再改成异步的。

2.异步回读

异步方式需要借助Pixel Buffer，这篇文章写得很好。OpenGL Pixel Buffer Object (PBO)，它的一幅插图就解释了整个流程，如下：

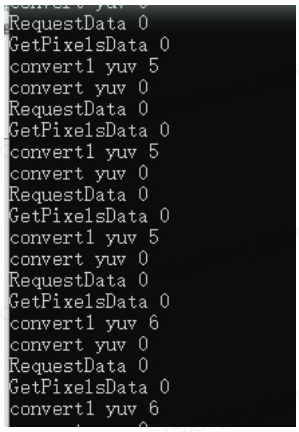


所以我的实现如下：

```
RequestData(pbo_index, render_width, render_height);
std::vector<unsigned char> data(render_width * render_height * 4, 0);
auto tmp = &data[0];
GetPixelsData(next_pbo_index, &tmp, render_width, render_height);
pbo_index = (pbo_index + 1) % pbo_cnt;
next_pbo_index = (next_pbo_index + 1) % pbo_cnt;
writer.Write(data, cnt * 33);
++cnt;
```

这里面的pbos就是pbo的数组，错开RequestData和GetPixelsData就可以减少阻塞时间，而RequestData和GetPixelsData的实现如下：

```
void RequestData(int pbo_index, int render_width, int render_height)
{
	auto start = std::chrono::steady_clock::now();

	glBindBuffer(GL_PIXEL_PACK_BUFFER, pbos[pbo_index]);
	glPixelStorei(GL_PACK_ALIGNMENT, 1);
	glReadPixels(0, 0, render_width, render_height, GL_RGBA, GL_UNSIGNED_BYTE, nullptr);
	CHECK_ERROR;
	glPixelStorei(GL_PACK_ALIGNMENT, 4);
	glBindBuffer(GL_PIXEL_PACK_BUFFER, 0);

	auto finish = std::chrono::steady_clock::now();
	auto ms = std::chrono::duration_cast<std::chrono::milliseconds>(finish - start).count();
	std::cout << "RequestData " << ms << std::endl;
}


void GetPixelsData(int pbo_index, unsigned char** data, int render_width, int render_height)
{
	auto start = std::chrono::steady_clock::now();

	glBindBuffer(GL_PIXEL_PACK_BUFFER, pbos[pbo_index]);
	const void* src_data = glMapBufferRange(GL_PIXEL_PACK_BUFFER, 0, render_width * render_height * 4, GL_MAP_READ_BIT);
	if (src_data)
	{
		memcpy(*data, src_data, render_height * render_width * 4);
	}
	glUnmapBuffer(GL_PIXEL_PACK_BUFFER);
	glBindBuffer(GL_PIXEL_PACK_BUFFER, 0);

	auto finish = std::chrono::steady_clock::now();
	auto ms = std::chrono::duration_cast<std::chrono::milliseconds>(finish - start).count();
	std::cout << "GetPixelsData " << ms << std::endl;
}
```



我突然想到，如果用一个Pbo呢？，于是我做了个实验。我改了一下RequestData，加了两行：

```
void RequestData(int pbo_index, int render_width, int render_height)
{
	auto start = std::chrono::steady_clock::now();

	glBindBuffer(GL_PIXEL_PACK_BUFFER, pbos[pbo_index]);
	glPixelStorei(GL_PACK_ALIGNMENT, 1);
	glReadPixels(0, 0, render_width, render_height, GL_RGBA, GL_UNSIGNED_BYTE, nullptr);

	const void* src_data = glMapBufferRange(GL_PIXEL_PACK_BUFFER, 0, render_width * render_height * 4, GL_MAP_READ_BIT);
	glUnmapBuffer(GL_PIXEL_PACK_BUFFER);

	CHECK_ERROR;
	glPixelStorei(GL_PACK_ALIGNMENT, 4);
	glBindBuffer(GL_PIXEL_PACK_BUFFER, 0);

	auto finish = std::chrono::steady_clock::now();
	auto ms = std::chrono::duration_cast<std::chrono::milliseconds>(finish - start).count();
	std::cout << "RequestData " << ms << std::endl;
}
```

然后，我仅仅调用RequestData，结果如下：

`glMapBufferRange` will force the `glReadPixels` call to finish if it wasn't (since you're now accessing the pointer on the CPU, there's no way around that).

So... don't do the 2 back-to-back, but rather significantly later.

这就是为啥用多个pbo错开时间，neox的Gaea分支的实现甚至用了6个pbo，就是为了更进一步错开readpixels和map的时间，尽可能消除阻塞。

3.对齐问题

大家知道内存对齐很常见，GPU当然也有，前面的代码里面很多glPixelStorei的调用，这就是为了调整GL_PACK_ALIGNMENT，默认为4。这个对齐很关键，不过本文仅仅讨论回读的对齐。
我做个实验，设置渲染尺寸为 221 * 181。
读取的时候：

```
glPixelStorei(GL_PACK_ALIGNMENT, 1);
glReadPixels(0, 0, render_width, render_height, GL_RGBA, GL_UNSIGNED_BYTE, *data);
glPixelStorei(GL_PACK_ALIGNMENT, 4);
```

很好，录制的视频如下：

完美，视频是ok的。

那么，我改一下对齐：

```
glPixelStorei(GL_PACK_ALIGNMENT, 8);
glReadPixels(0, 0, render_width, render_height, GL_RGBA, GL_UNSIGNED_BYTE, *data);
glPixelStorei(GL_PACK_ALIGNMENT, 4);
```

大家觉得会咋样。
gl会报错，报错为`GL error 1282 at line 266`，也就是这里：

```
glReadPixels(0, 0, render_width, render_height, GL_RGBA, GL_UNSIGNED_BYTE, nullptr);
CHECK_ERROR;
```

为啥呢，因为pbo的内存大小不够了，越界了，所以这里出了问题。我们算一下，
221 * 4 = 884 ,它不是8的倍数，而我们要求8的对齐，所以GPU回读的时候，一行变成了888，888 * 181当然就比pbo大了，我们改一下pbo的大小：

```
glBufferData(GL_PIXEL_PACK_BUFFER, (render_width + 1) * render_height * 4 , nullptr, GL_DYNAMIC_READ);
```

如上，相当于每一行加了4，也就是884变成了888，这就gl就不会报错了。
但是视频变成这样了，如下图：

像是被剪切了，这是为啥呢？
这是因为数据的内存布局变了，数据量也变了，要这样改：
首先数据量变了，改一下内存大小，每一行加1即可：

其次，memcpy改一下大小：

```
const void* src_data = glMapBufferRange(GL_PIXEL_PACK_BUFFER, 0, render_width * render_height * 4, GL_MAP_READ_BIT);
if (src_data)
{
    memcpy(*data, src_data, render_height * (render_width + 1) * 4);
}
```

最后，我们转换的时候，改一下stride，如下：

```
const int src_stride[] = { static_cast<int>(m_width * 4 + 4), 0, 0, 0 };
int ret = sws_scale(m_sws_context, &surface_data, src_stride, 0, m_height, av_frame_ptr->data, av_frame_ptr->linesize);
```

好，这样就ok了，这样也太麻烦了，而且容易出错。所以我选择1对齐，紧凑布局。这样虽然会有一点点性能损失，当我觉得我宁愿损失这点性能，上面不为1的调整也太麻烦了。
当然，其实上面的例子是我凑出来的，一般视频编码不会出现，因为首先RGBA就是4字节，其次视频宽高一般都是偶数，这样不就已经8对齐了吗？而GL_PACK_ALIGNMENT取值最大也就是8。但是为了稳妥，我还是先设置成1，然后restore为默认值4。

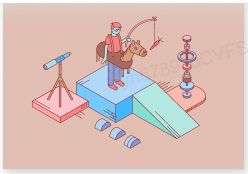
4.总结

录制视频这种频率高的回读操作，还是必须用异步的，同步会浪费CPU时间，降低帧率。

肖宏宇 编 44 4个资源

音视频开发的相关实践和研究记录

最近更新：回读GPU缓存数据 同步VS异步



大家都在看

【NeoX引擎】cocos ui加载优化

UE5 Nanite 浅析 (一) : 核心思路

UE引擎各类pipeline（超清大图）